УДК 004.424, 004.415

В.Г. Колесник

DS-ТЕОРИЯ КАК ПРОТОТИП ТЕОРИИ ПРИКЛАДНЫХ АЛГОРИТМОВ

Предложено и описано понятие схемы декомпозиции как ядра *DS*-теории. Описаны понятия свойства-совокупности и алгоритмической зависимости. Предложено использовать схему декомпозиции, учитывая ее алгоритмическую природу, как основу для построения универсального (канонического) алгоритма. Обсуждаются вопросы преобразования канонического алгоритма в конкретные прикладные алгоритмы. Описываются операции над схемами декомпозиции, которые являются и операциями над алгоритмами.

Введение

В работе предлагается описание теоретической модели, называемой схемой декомпозиции (decomposition schemes *DS*). Схема декомпозиции является неотъемлемой частью теории схем декомпозиции (*DS*-теории). *DS*-теория рассматривается как теория или как прототип теории для проектирования прикладных алгоритмов. В подобной теории нуждается технология программирования [1, 2].

Созданию теоретической модели и *DS*-теории предшествовало использование метода проектирования программ по структуре данных JSP [3], Logical Construction of Programs [4] и R-технология [5] при разработке программного обеспечения. В какойто момент ряд исследователей независимо друг от друга обратили внимание на то, что структура обрабатываемых данных может определять структуру программы. Класс таких программ достаточно большой. Подобную зависимость увидели много тех программистов-практиков, то разрабатывал программы для обработки линейных файлов и иерархических баз данных. Принципиальным недостатком метода было то, что не существовало ясного и детального способа определить структуру данных.

В результате поиска автором было обнаружено, что именно схема декомпозиции определяет структуру данных и опосредованно определяет структуру программы. Таким образом, основные идеи *DS*-теории появились в результате попыток развить метод программирования по структуре данных с целью расширения сферы его применения [6] и с целью более глубокого обоснования самого метода [7]. Схема декомпозиции в методическом плане ответила на вопросы проектирования алгоритмов обработки данных 70-х и 80-х. Но потенциальные возможности *DS* как средства проектирования алгоритмов значительно шире и поэтому она положена в основу описываемой далее *DS*-теории.

Цели создания и развития *DS*-теории следующие:

– уменьшение сложности алгоритмов за счет системы более емких понятий;

– уменьшение сложности алгоритмов за счет создания графических средств проектирования алгоритмов;

– создание принципов генерации алгоритмов.

Понимание важности емких понятий для технологии программирования есть в научном сообществе [8]. Реализацией этого подхода – использование емких понятий – было создание алгоритмических языков, макросов, механизма подпрограмм. Емкость макроса или оператора алгоритмического языка больше чем емкость машинной команды или строки мнемокода. Тем не менее, практически одновременно с изобретением первых алгоритмических языков пришло понимание того, что выразительные возможности этих языков весьма ограничены [9].

Позже были созданы специализированные инструменты – генераторы отчетов и языки моделирования. Здесь тоже использовались понятия более емкие, чем оператор алгоритмического языка или макрос. Но в дальнейшем развития технологии программирования в этом направлении не происходило. Понятиями *DS*-теории, обладающими большой емкостью, являются: совокупное свойство (С-свойство), алгоритмическая зависимость (А-зависимость), схема декомпозиции и контур синтеза (КС).

Основная парадигма *DS*-теории

Принципиальная ключевая идея теории (проектная парадигма) звучит так: "В компьютере производятся новые знания". Эта парадигма принимается в теории как аксиома, как исходное положение и противопоставляется другой парадигме: "В компьютере происходит обработка информации". Развитие основной идеи в следующем: "Знания имеют свою внутреннюю структуру и свой механизм их формирования – процедуру декомпозиции". И далее: "Алгоритм и программа в компьютере являются отражением процедуры декомпозиции ".

Процедура декомпозиции известна и интуитивно понятна практически любому исследователю, так как это один из мыслительных инструментов исследований. Кроме того, схема декомпозиции – это то, что очень похоже на структурный анализ [10]. Но в *DS*-теории акценты устанавливаются на процедуре формирования знаний, а не на механизме декомпонуемых конструкций и управлении ими. Процедура декомпозиции – это реализация проектной парадигмы.

Ожидаемая эффективность и продуктивность *DS*-теории в том, что в ее рамках разрабатываются и исследуются принципы формирования алгоритмов. Алгоритм выводится задолго до начала программирования и даже вне программирования. *DS*-теория ориентирована на устранение из процесса проектирования алгоритмов и программ личностного субъективного фактора.

Свойства объекта

Анализ и исследования в *DS*-теории отталкиваются от идеи "объект"*. Объектами могут быть реальные объекты, существующие в окружающей человека действительности. Естественные и искусственные. Естественные: географический участок поверхности Земли, минерал, человеческий организм и т. п. Искусственные: изделие машиностроения или легкой промышленности и т. п. Как объект может рассматриваться нечто статическое, относительно неизменное и обозримое, например, стол, дом или автомобиль. Как объект может рассматриваться и нечто такое, что не имеет видимой формы, например, устный договор между субъектами предпринимательской деятельности. Как объект могут рассматриваться различного вида процессы. Например, процесс пищеварения или наблюдение за экспериментом.

Выделив несколько объектов из окружающей действительности, можно рассматривать их как систему. В рамках системы каждый объект в отношении с каким-либо другим объектом проявляет свои свойства. Свойства объекта существуют независимо от других объектов. Собственно, свойство – это проявление внутренней структуры объекта.

Свойства объектов для человека проявляются тогда, когда объекты вступают в какие-либо отношения (высота стола = 75 см. проявилась в сравнении

* Понятие "объект" используется в общенаучном и философском значении и не имеет отношения к понятию "объект" в объектно-ориентированном проектировании и программировании.

с другим столом, с высотой человека, с мерной линейкой). Эти отношения могут быть реальными или мыслимыми человеком, когда он соотносит и сравнивает эти объекты в своем сознании.

Свойства объекта или его частей представляются формулой или квалификационным предложением: “Субъект есть предикат”. Квалификационное предложение может быть трех видов.

Первый вид – простейший, отражает конкретное значение одного свойства или одной характеристики конкретного объекта. Пример: Объект – деталь. Свойство объекта – вес. Квалификационное предложение имеет вид: “1214 – 15 кг”. Здесь “1214” номер детали, а “15 кг” – вес детали. Этот вид квалификационного предложения называется элементарным свойством (Эл-свойством).

Второй вид квалификационного предложения отражает конкретные значения нескольких свойств одного конкретного объекта (или одной части). Пример: Объект – деталь. Свойства объекта – вес, материал, количество операций, суммарная стоимость обработки. Квалификационное предложение имеет вид: “1214 – 15 кг, сталь 40, 8, 35”. Здесь “1214” номер детали, а “15 кг.” – вес детали, “сталь 40” – материал, “8” – количество операций для изготовления детали, “35” – количество нормо-часов для изготовления детали. Этот вид квалификационного предложения называется агрегированным свойством (А-свойством).

Третий вид квалификационного предложения обусловлен ситуацией, когда в качестве свойства объекта рассматривается перечень свойств его компонент. Такой перечень – совокупное свойство (С-свойство).

Пример. Характеристикой предприятия железобетонных изделий может быть каталог изготовляемых изделий:

- плита перекрытия;
- кольцо колодца;
- лестничный марш;
- лестничная площадка и т. д.

Характеристикой того же предприятия наряду с интегральными показателями, выраженными одним числом – квартальный объем реализации или квартальная прибыль, может быть перечень реализованных изделий с указанием объемов реализации. Квалификационное предложение в таком случае имеет вид:

объем реализации продукции в первом квартале:

- плита перекрытия 120 шт.;
- кольцо колодца 70 шт.;
- лестничный марш 250 шт.;
- плита перекрытия 230 шт. и т. д

Здесь объект – квартальный объем реализации продукции. Номер объекта – первый. Свойство объекта – перечень продукции, изготовленной в первом квартале.

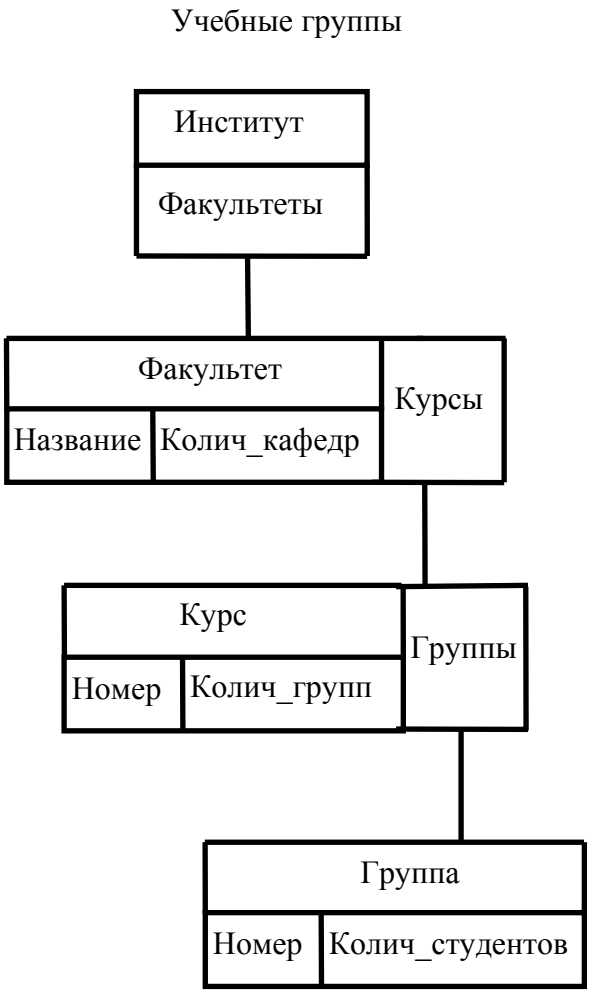
Совокупным свойством объекта может быть просто перечень названий или номеров компонент. Свойство объекта может в таких случаях быть скрыто в самом названии или в номере компоненты.

С-свойство вместе Эл-свойством и А-свойством имеют объемлющее их родовое понятие “свойство” (Р-свойство). В *DS*-теории наряду со свойствами объекта рассматриваются также данные об объекте. Данное об объекте (Р-данное) или данная величина, характеризующая объект, – это свойство объекта, для которого определен знак, читаемый или воспринимаемый человеком или (и) техническими средствами. Аналогично понятиям С-свойство, Эл-свойство и А-свойство существуют понятия: совокупное данное (С-данное), агрегированное данное (А-данное) и элементарное данное (Эл-данное). Р-свойства и Р-данные соотносятся как с объектом, так и с частями объекта. А-данные могут включать наряду с Эл-данными А-данные и С-данные. С-данные могут содержать А-данные.

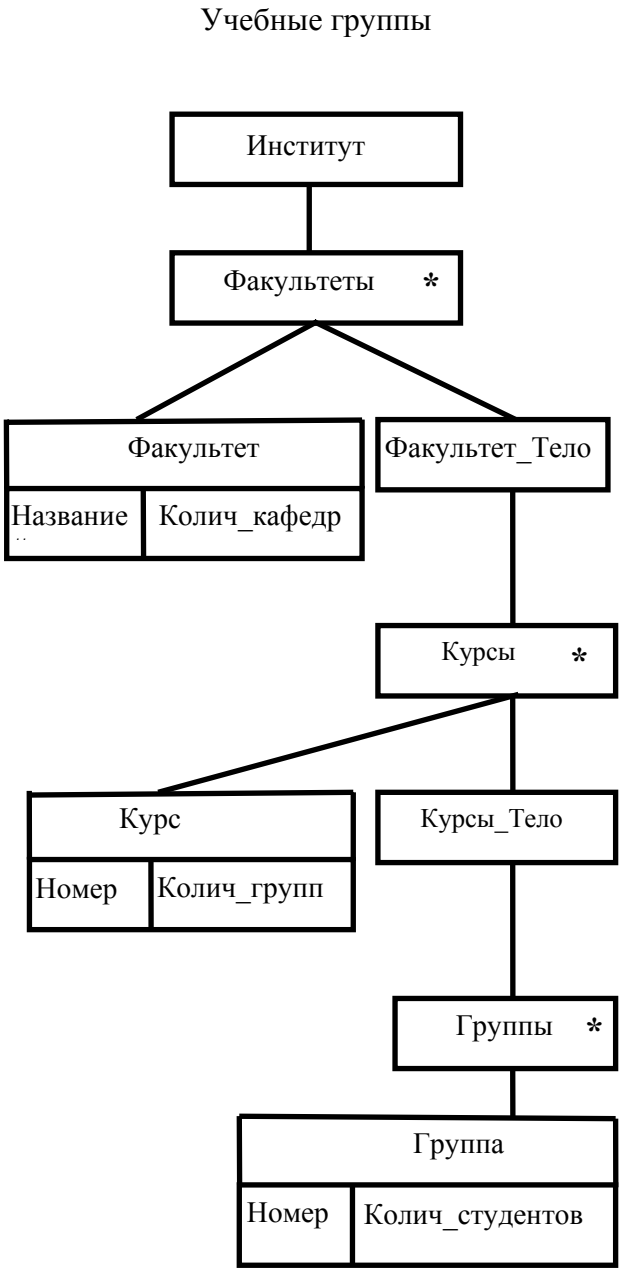
В результате процедуры декомпозиции может быть получено некоторое

число частей (компонент). Если для каждой части определено числовое значение одного и того же свойства, например, вес части, то совокупность всех этих значений рассматривается как свойство одного типа (или данное одного типа). Типы данных и типы свойств именуются. Для изображения всех типов P-данных (а также типов P-свойств), получаемых в результате декомпозиции, используется древовидный граф.

На рис. 1 показанно сравнение нотации для изображения типов P-данных в сравнении со способом нотации структур данных, принятом в JSP [2]. Это сравнение правомерно, так как сопоставляются одни те же явления.



а



б

Рис. 1. Представление одного и того же информационного объекта различными видами графики:

а – графика DS-теории (Факультеты, Курсы, Группы – C-данные (C-свойства));
б – графика метода JSP

Алгоритмическая зависимость

Другое ключевое понятие *DS*-теории – алгоритмическая зависимость (А-зависимость) между свойствами объекта.

Отличие А-зависимости от функциональной зависимости в том, что для описания алгоритмической зависимости может быть использовано наряду с формулами и математическими знаками также текстовые описания и различные табличные конструкции. Алгоритмическая зависимость полностью формально может быть записана только с помощью средств алгоритмического языка.

Рассматриваются две категории зависимостей.

Первая категория – аналитическая зависимость (A^a-зависимость) отражает зависимости между Эл-свойствами и (или) А-свойствами объекта (или одной и той же части объекта).

Вторая категория – синтетическая зависимость отражает зависимость (A^s-зависимость) между Р-свойствами объекта и Р-свойствами частей, непосредственно составляющих объект. Это также может быть зависимость между Р-свойствами части объекта и Р-свойствами частей, непосредственно составляющих данную часть объекта. Вторая категория включает два вида A^s-зависимостей.

Первый вид – это ситуация, в которой аргументом является Эл-свойство объекта, а функцией – Эл-свойство частей объекта или в другой нотации:

$$R = \alpha\ (Q),$$

где Q – Эл- свойство объекта, R – Эл-свойство компоненты объекта, а α – обозначение A^s-зависимости. Если С-свойство P состоит из Эл-свойств R ($P = (\cup R)$), то возможна еще и такая запись:

$$P = \alpha\ (Q).$$

Второй вид – это ситуация, в которой аргументом является Р-свойства частей объекта, а функцией – Р-свойства объекта или в другой нотации:

$$W = \gamma\ (V),$$

где W – Эл-свойство объекта, V – С-свойство части объекта, а γ – название А-зависимости. Если С-свойство Z состоит из Эл-свойств V ($Z = (\cup V)$), то возможна еще и такая запись:

$$W = \gamma\ (Z).$$

Представителем этого вида A^s-зависимости может быть вычисление суммы значений, поиск минимального или максимального элемента, любой расчет или любое вычисление, в котором обрабатываются все элементы некоторой совокупности.

Реализация А-зависимостей

Для изображения того, как алгоритмически реализуются А-зависимости и для изображения алгоритма, который может быть построен на основе схемы

декомпозиции, используется гипотетическое вычислительное устройство*, состоящее из входной и выходной лент, процессора и ленты, содержащей описание А-зависимостей.

Заданные (известные) свойства объекта, представленные как Эл-данные, размещены на входной ленте.

Искомые свойства объекта, представленные как Эл-данные, получаемые в результате расчета, размещаются на выходной ленте. Одно Эл-данное размещается в одной ячейке** ленты, а А-данные и С-данные занимают группы ячеек. К входной ленте имеет доступ читающая головка, которая может прочитать содержимое только одной из ячеек.

Описание А-зависимостей, в соответствии с которыми определяются искомые свойства, размещаются на ленте процессора. Описание одной А-зависимости размещается в одной ячейке. Входные данные и соответствующие им описания А-зависимостей размещаются на соответствующих им лентах синхронно.

Вычислительное устройство реализует обработку данных (вычисление Р-свойств) в соответствии с теми А-зависимостями, которые есть между Р-свойствами. Устройство действует следующим образом. Обе ленты движутся справа налево. В каждый момент времени доступна только одна ячейка для чтения и одна ячейка для записи.

Аналогично, описание А-зависимости доступно для чтения читающей головкой процессора. Возможна такая ситуация, в которой при обработке одного Эл-данного будет реализовано несколько А-зависимостей.

Для следующего Эл-данного снова будет повторена эта группа А-зависимостей. Графически эта группа размещается в прямоугольнике. Прямоугольник – это зона процессора.

Начиная обработку очередного Эл-данного, читающая головка в пределах зоны процессора будет регулярно возвращаться к первому из описаний и А-зависимостей. Аналитическая A^a-зависимость алгоритмически реализуется следующим образом (рис. 2, а).

Процессор в соответствии с описанием А-зависимости β обрабатывает Эл-данное (D_i), которое доступно на входной ленте. Результат обработки Эл-данное C_i записывается в ячейку на выходной ленте. Обработка завершается тогда, когда на входной ленте исчерпаны Эл-данные.

Синтетическая зависимость первого вида алгоритмически реализуется следующим образом (рис. 2, б).

Процессор в соответствии с описанием А-зависимости α обрабатывает Эл-данное R, которое доступно на входной ленте. Результат обработки С-данное Q записывается в группу ячеек на выходной ленте. Входная лента и лента процессора не движутся, выходная лента смещается по мере заполнения ячеек Эл-данными Q_i. Завершение обработки определяется характером синтетической зависимости.

Синтетическая зависимость второго вида алгоритмически реализуется следующим образом (рис. 2, в).

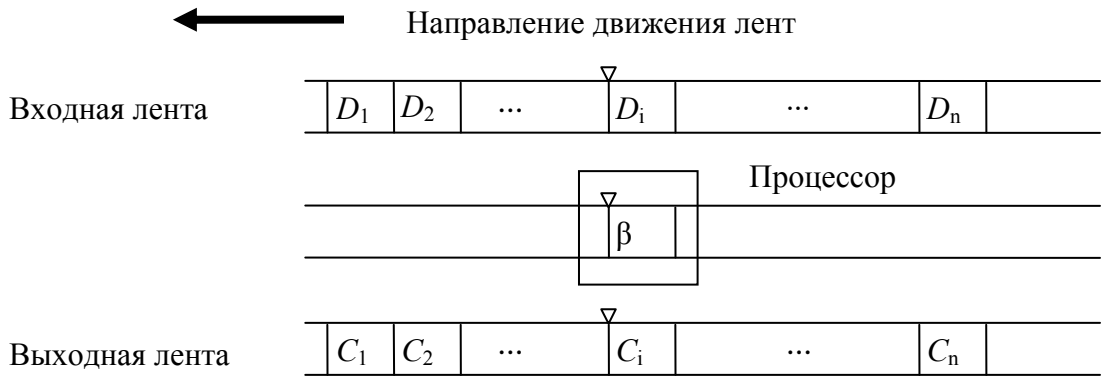
Процессор в соответствии с описанием А-зависимости γ обрабатывает С-данное V, которое доступно на входной ленте. Результат обработки Эл-данное W записывается в ячейку на выходной ленте.

Выходная лента и лента процессора не движутся, входная лента смещается по мере чтения ячеек с Эл-данными V_i. Завершение обработки определяется характером синтетической зависимости и размером С-данного V.

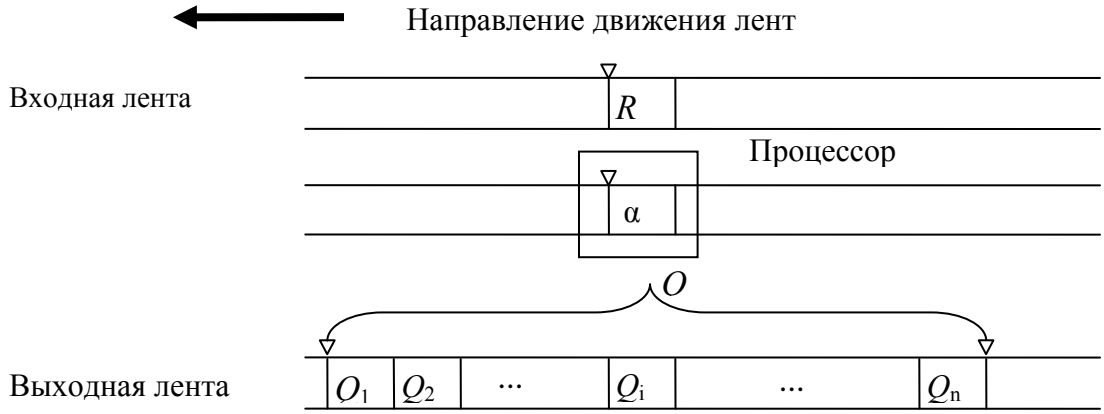
---

* Вычислительное устройство похоже на алгоритмическую систему “машина Тьюринга”. Различие в том, что в ячейках входной и выходной лент размещаются Эл-данные, вместо управляющего устройства используется понятие “процессор”, а алгоритм определяется содержимым ленты процессора, а не внутренними состояниями устройства управления.

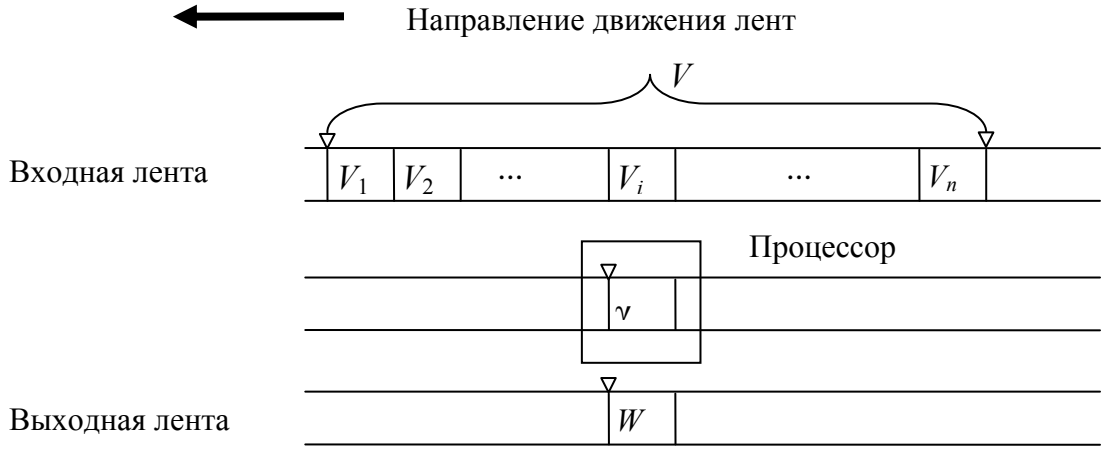
** Предполагается, что размеры данных совпадают с размерами ячеек, хотя в практике электронной обработки данных это не так.



а



б



в

Рис. 2. Алгоритмы реализации алгоритмических зависимостей:
а – A^a-зависимости $C= \beta (D)$. C и D – Эл-данные;
б – A^s-зависимости $Q= \alpha (R)$. Q – С-данное, R и Q_i – Эл-данные;
в – A^s-зависимость $W= \gamma (V)$. W и V_i – Эл-данные. V – С-данное

Процедура декомпозиции

В наиболее общем виде познавательный процесс – это движение от явления к сущности, – это движение от созерцания формы объекта к пониманию его содержания. Такое движение – это движение от явных, более доступных восприятию свойств объекта к его сущностным свойствам. Такое движение характерно тем, что это есть движение от легче определяемых характеристик объекта к труднее определяемым его характеристикам.

Если между парой характеристик или свойств объекта, одна из которых отражает форму, а другая – содержание объекта, существует зависимость и она известна, то, используя ее и значение первой можно определить значение второй. Если такая зависимость неизвестна, то объект рассматривается как система или как нечто составное и выполняется следующее:

а) объект делится на части;

б) для каждой части определяется пара характеристик (формы и содержания) и существующая между ними зависимость;

в) по конкретным значениям формальных характеристик определяются содержательные характеристики частей первого уровня декомпозиции;

г)на основе полученных сведений о частях синтезируется значение содержательной характеристики объекта.

Таким образом, выполнена процедура декомпозиции. Для этого механизма формирования знаний несущественно, каким образом строятся схемы декомпозиции (схемы деления) объекта и схемы синтеза сведений (свойств, данных), также несущественно, каким образом ведется поиск зависимостей между характеристиками. Важно то, что в результате поиска, зависимость может быть найдена или нет. Если зависимость не найдена, то для ее поиска выполняется декомпозиция более глубокого уровня, а также выполняются и все остальные пункты – б, в, г. Более мелкое (более глубокое) членение частей объекта будет продолжаться до тех пор, пока на очередном уровне не будет найдена зависимость между характеристиками и с помощью этой зависимости не будут найдены свойства этих наименьших частей. Затем с помощью этих свойств будут найдены свойства частей их составляющих (частей предшествующего уровня). Процедура синтеза, в процессе движения от свойств включаемых частей к свойствам включающих, будет завершена, когда будет определена искомая характеристика (искомое свойство) исходного объекта.

Декомпозиция может быть остановлена, если выясняется бесперспективность ее продолжения. Это значит, что нет способа для последующего более глубокого деления частей или после очередного деления не находятся продуктивные зависимости между характеристиками полученных частей или не существует процедуры синтеза характеристик для включающего уровня. Декомпозиция имеет ценность, если после ее завершения происходит генерация данных о самом объекте – синтез данных.

Различаются частные схемы и полные схемы декомпозиции. Частная схема декомпозиции (схема декомпозиции одного уровня) показывает то, как разделить объект на непосредственно составляющие его компоненты. Частная схема декомпозиции должна определять то, как выделяется каждая из компонент. Частная схема может быть использована для деления как объекта, так и его части. Полная схема декомпозиции – это результат декомпозиции объекта после приложения всех понадобившихся в процессе деления частных схем.

Одна и та же схема одного уровня декомпозиции может применяться много раз – для объектов различных типов. То есть, могут существовать универсальные процедуры декомпозиции.

Все части, полученные в результате применения частной схемы декомпозиции, могут быть сгруппированы. Признаки, в соответствии с которыми, части включаются в группы, определяет исследователь. Группа частей называется типом части. В схеме декомпозиции каждая часть приписывается некоторому типу. На каждом уровне декомпозиции порождается собственный набор типов. Набор включает хотя бы один тип. В DS-теории как результат частичной или полной декомпозиции рас-

сматривается именно набор типов. В результате реальных декомпозиций возможны различные количества частей принадлежащих определенному типу. В зависимости от объекта или (и) схемы декомпозиции может быть получено произвольное количество частей одного типа (экземпляров типа), хотя бы одна часть, одна часть обязательно, ни одной части и т. п. В результате декомпозиции объекта может получено произвольное число частей, типов частей и уровней.

Графически результат декомпозиции может быть представлен деревом. В общем виде схема декомпозиции показана на рис. 3. На этом дереве каждый узел соотносится с одним типом части. Дерево может содержать произвольное количество уровней, узлов, ребер. Каждый тип части может иметь произвольное количество Р-свойств и А-зависимостей между Р-свойствами. А-зависимости соотносятся только с типами частей (с узлами дерева). Прежде всего, дерево отображает иерархию типов частей объекта, но может также отображать типы свойств, А-зависимости и частные схемы декомпозиции. Полная схема декомпозиции показывает возможность деления типов частей по определенным частным схемам, но не отображает детальный порядок деления частей и вычисления в соответствии с А-зависимостями.

С точки зрения дерева, показанного на рис. 3, декомпозиция это движение от корневого узла к листьям – сверху вниз. Синтез данных, определение новых сведений об объекте может быть частично или полностью материальным – взвешивание, определение количества бракованных компонент или замена поломанной детали. Синтез данных это движение от листьев к корневому узлу – снизу вверх.

Описание схемы декомпозиции содержит дерево полной схемы декомпозиции (*DPS*), где для каждого узла K_i должен быть определен кортеж описания узла $KR_i = (T_i, A^a_i, A^s_i, DM_i)$, который включает:

- T_i – перечень типов свойств.
- A^a_i – аналитические зависимости.
- A^s_i – синтетические зависимости.
- DM_i – частная схема декомпозиции.

K_i – идентификатор узла, где индекс

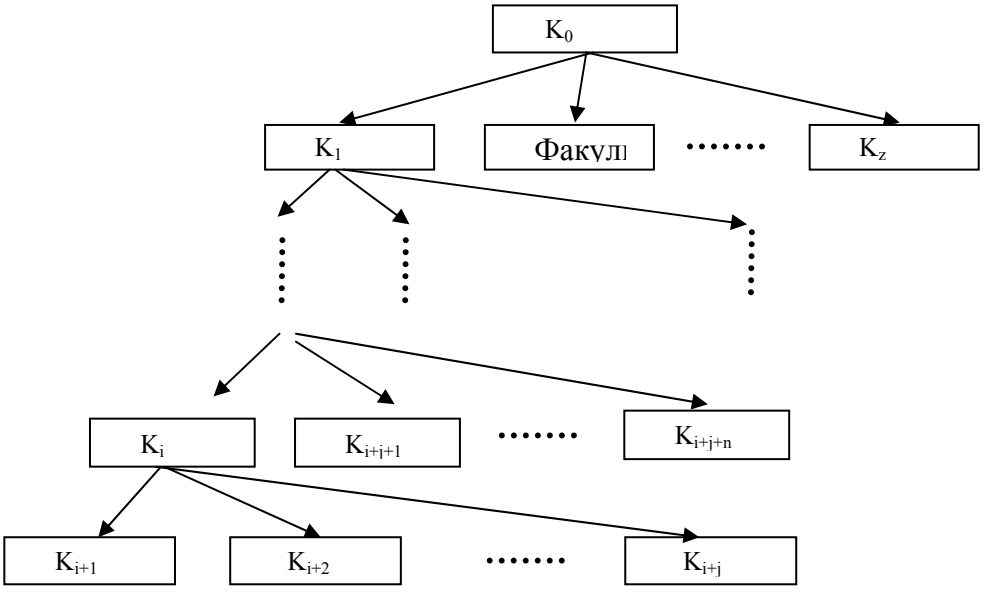


Рис. 3. Дерево схемы декомпозиции объекта. Индекс – номер узла. Уровни декомпозиции пронумерованы с нулевого по N-й

отражает номер узла при обходе дерева сверху вниз слева направо.

Символическое описание схемы декомпозиции имеет вид:

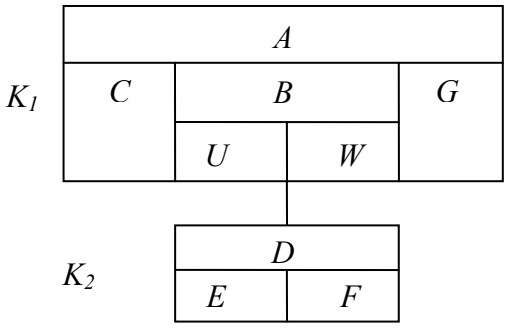
$DPS=\{K_0[KR_0]\ (K_1[KR_1]\ (\ldots),\ K_r[KR_r]\ (\ldots),\ldots,K_z[KR_z]\ (\ldots))\}$.

Кортежи листьев не содержат частные схемы декомпозиции и синтетические зависимости.

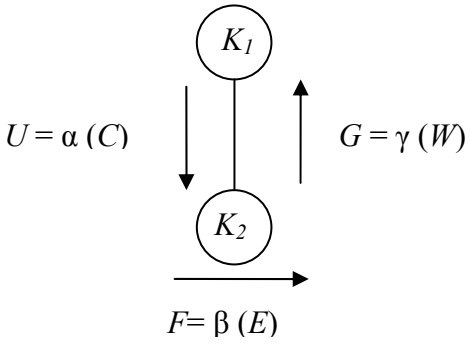
Схема декомпозиции и канонический алгоритм

Схема декомпозиции в чистом виде не является алгоритмом. Схема декомпозиции показывает только принципиальную возможность того, как можно выполнить декомпозицию, но не описывает детально порядок ее выполнения. Для того чтобы процедура декомпозиции превратилась в алгоритм, необходимо определить порядок доступа к Р-данным и порядок их обработки (порядок реализации А-зависимостей).

В качестве примера на рис. 4, а показана схема декомпозиции. В результате выполненной декомпозиции получено N частей. Дерево декомпозиции содержит два узла, которым присущи С-свойства: U и W. $U = \cup E_N$, $W = \cup F_N$. На рис. 4, б показана та же схема декомпозиции с присущими ей А-зависимостями*. С узлом *K1* соотносятся А-зависимости $U = \alpha\ (C)$ и $G = \gamma\ (W)$. А с узлом *K2* соотносится А-зависимость $F = \beta\ (E)$.



а



б

Рис. 4. Схема декомпозиции:
а – структура свойств объекта;
б – контур синтеза свойств объекта

Реализация этой схемы заключается в выполнении трех шагов.

А-зависимость $U = \alpha\ (C)$ – это синтетическая зависимость первого вида. Для ее реализации необходимо использовать алгоритм, показанный на рис. 2, б. Результат выполнения этого алгоритма – Эл-данное $U = \cup E_i$, где i меняется от 1 до N.

А-зависимость $F = \beta\ (E)$ – это аналитическая зависимость. Для ее реализации необходимо использовать алгоритм, показанный на рис. 2, а. Результат выполнения этого алгоритма – Эл-данное $W = \cup F_i$, где i меняется от 1 до N. Каждое Эл-данное E_i является исходным для вычисления Эл-данного F_i.

А-зависимость $G = \gamma\ (W)$ – это синтетическая зависимость второго вида. Для ее реализации необходимо использовать алгоритм, показанный на рис. 2, в. Результат выполнения этого алгоритма – Эл-данное G. Для вычисления G было использовано каждое Эл-данное F_i.

В процессе выполнения этих трех шагов потребуется два раза возвращать входные ленты в исходное положение. Окончательно для реализации схемы декомпозиции и синтеза новых значений искомых свойств потребуется три прохода. Вычисления этих трех проходов можно совместить и реализовать с помощью алгоритма, показанный на рис. 5. В алгоритме с каждым продвижением ленты на одну

* На рис. 4 и на некоторых следующих рисунках А-зависимости соотносятся с ближайшими узлами, расположенными выше.

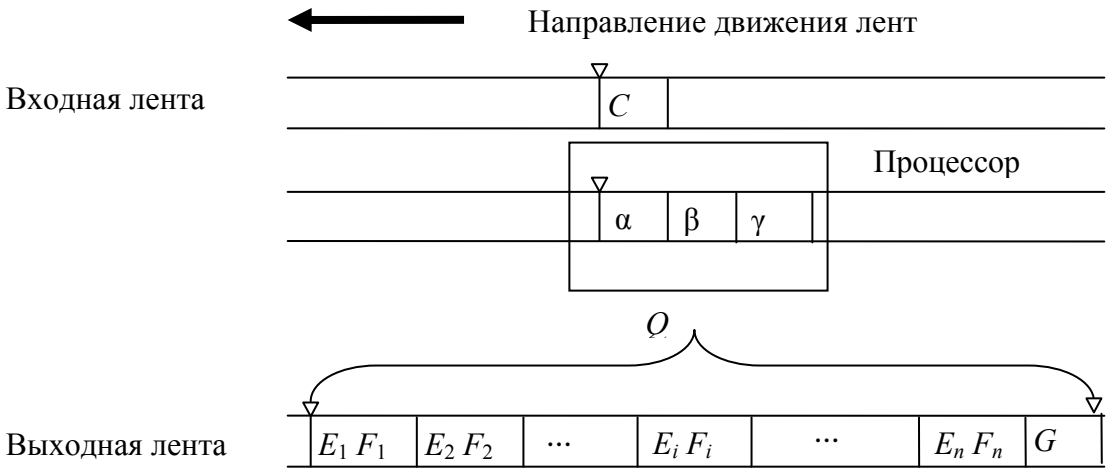


Рис. 5. Алгоритм, реализующий полную схему декомпозиции одного уровня

ячейку выполняются три операции:

- вычисляется Эл-данное E_i и записывается в ячейку i;
- вычисляется Эл-данное F_i и записывается в ячейку i;
- выполняется формирование С-данного G.

Вслед за формированием экземпляра E_N формируется экземпляр F_N и завершается формирование Эл-данного G.

Эта вычислительная процедура называется алгоритмом реализации DS. Последний демонстрирует алгоритмическую природу DS в традиционном для программирования смысле. Алгоритм реализации DS называется также каноническим алгоритмом.

Перечень всех А-зависимостей, составленный в процессе обхода дерева декомпозиции сверху вниз и слева направо, называется контуром синтеза свойств объекта или просто контуром синтеза. В условиях предыдущего примера контур синтеза представляет собой перечень названий А-зависимостей: $U = \alpha\ (C)$; $F = \beta\ (E)$; $G = \gamma\ (W)$ или α -› β -› γ. С учетом этого определения имеет место следующее утверждение: “Канонический алгоритм реализует КС”.

Реальные схемы декомпозиции имеют произвольное количество уровней декомпозиции, как правило, более двух. С каждым узлом соотносится произвольное количество А-зависимостей. Из А-зависимостей разных уровней могут быть составлены цепочки трех разных видов. Цепочки первого вида состоят из синтетических зависимостей первого вида. Они описывают процедуру декомпозиции объекта или его частей и процедуры вычислений свойств включаемых компонент на основании свойств включающих их частей объекта. Цепочки второго вида состоят из синтетических зависимостей второго вида. Они описывают процедуру синтеза свойств частей объекта или самого объекта на основании свойств включаемых компонент. Цепочки третьего вида включают А-зависимости между Эл-свойствами одного узла дерева *DS*. Из цепочек этих трех видов как из фрагментов и составляется КС любой схемы декомпозиции.

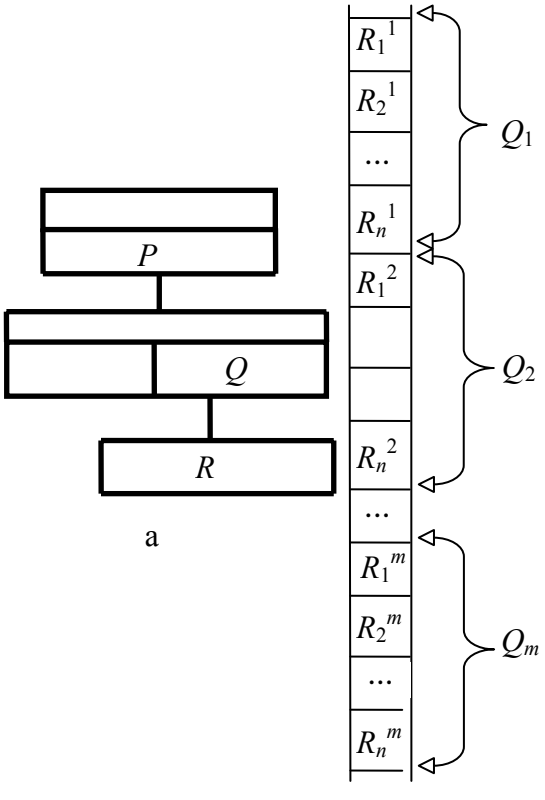
То, что *DS* имеют много уровней декомпозиции (рис. 6, а) предполагает, что С-данные могут содержать в качестве компонент другие С-данные (рис. 6, б). Вложенность групп С-данных может быть произвольной. Обработка вложенных групп С-данных в каноническом алгоритме порождает повторения вычислений, которые тоже вложенные друг в друга. Вложенные повторения вычислений изображаются в традиционных строчных языках программирования с помощью вложенных циклов (рис. 6, в, г). Тот факт, что повторения (и соответственно циклы) могут быть вложенными, тоже изображается деревом (рис. 6, д). Так называемое дерево

циклов.

Именно дерево циклов – основной каркас большинства прикладных реальных алгоритмов. Именно дерево циклов является существенным фактором сложности алгоритмов. Двумерный древовидный граф с десятками узлов вполне может быть размещен на листе бумаги формата А4 или на экране дисплея. А соответствующая этому графу программа, даже если из нее извлечь функциональность, займет в несколько раз больше пространства. Понятийная емкость дерева циклов и контура синтеза, как атрибутов теории *DS*, и есть преимуществом этой теории.

Схемы декомпозиции могут порождать также альтернативы, которые отражаются в каноническом алгоритме. Существование альтернатив обусловлено тем, что *DS* может содержать части разных типов, которые являются компонентами одной и той же включающей их части объекта.

Канонический алгоритм определяется как теоретическое, так и практическое явление, но он отличается от алгоритмов, существующих в реальном программном обеспечении. Это обусловлено тем, что в



а

б

DO WHILE …

DO WHILE …
…
END
Тело цикла. Обработка *R*
Тело цикла. Обработка *Q*
END

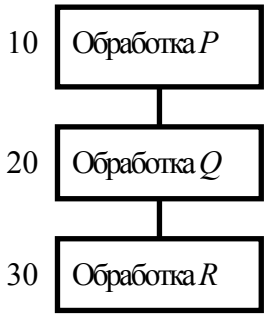
в

10. PERFORM 20 UNTIL

*Тело цикла. Обработка *Q*
20. PERFORM 30 UNTIL …

*Тело цикла. Обработка *R*
30. …

г



д

Рис. 6. *DS* и традиционные алгоритмические языки:

а – соподчиненность С-данных;

б – вложенность С-данных на входной ленте;

в – вложенные циклы для обработка С-данных, записанные с помощью алгоритмического языка структурного типа;

г – вложенные циклы для обработка С-данных, записанные с помощью коболоподобного алгоритмического языка;

д – графическое изображение вложенных циклов

реальных алгоритмах реализовано множество факторов связанных с конкретной вычислительной средой, с размещением данных и доступом к данным. Канонический алгоритм никаким образом не связан с каким-либо конкретным компьютером и с какой-либо конкретной вычислительной средой.

Алгоритмически релевантные факторы

От *DS* на бумаге или в сознании до конкретной прикладной программы должны быть сделаны ряд шагов. Эти шаги должны адаптировать канонический алгоритм к реальным условиям обработки данных.

Причины этого следующие:

- алгоритмическая модель может быть сложной из-за того, что входных лент* может быть более одной. Также и выходных лент может быть более одной;
- размер реальных С-данных, как правило, не совпадает с размерами ячеек на входных и на выходных лентах. Из-за этого необходимо включать в алгоритм блоки форматирования С-данных;
- модель последовательного доступа к С-данным самая простая. В практике электронной обработки данных есть более сложные методы доступа к данным;
- С-данные могут поступать не с ленты, а в режиме ONLINE от различных датчиков или в режиме диалога;
- С-данные могут быть выданы не только на ленту, но на монитор, на различные управляющие устройства;
- С-данные при размещении могут кодироваться, упаковываться и т. п.

Кроме этих причин существуют и другие. Каждая из перечисленных причин порождает группу факторов, которые воздействуют на канонический алгоритм. Эти факторы называются алгоритмически релевантными факторами. Для их реализации в канонический алгоритм добавляются алгоритмические конструкции. Из-за этого алгоритм усложняется и становится более тяжелым для понимания. Канонический алгоритм может превратиться в реальную программу только после того, как будут учтены и реализованы все группы факторов, обусловленные конкретной вычислительной средой. Каждая группа факторов влечет свой набор алгоритмических конструкций или свой алгоритмический слой, который накладывается на канонический алгоритм. Это так называемые адаптационные слои.

Схема декомпозиции, как и канонический алгоритм, являются инвариантом, неизменяющимся ядром прикладной программы с точки зрения условий реализации этой программы в различных средах вычислений. Схема декомпозиции и канонический алгоритм остаются неизменными независимо от того, на каком языке реализована программа, под управлением какой операционной системы выполняется программа, под управлением какой СУБД происходит доступ к данным.

Схема декомпозиции и канонический алгоритм порождают в анализе и проектировании те трудности, которые названы Бруксом [11] существенными (essence). Программирование адаптационных слоев порождает трудности ненаследуемые [11] или привносимые (accidents) в программное обеспечение.

В некоторых случаях схема декомпозиции может быть изображена только деревом, узлы которого нагружены именами С-данных. Этот граф можно назвать структурой данных*. Если С-данные размещены на одной входной ленте и на одной выходной, а также известен порядок их размещения на лентах, то этих сведений достаточно, чтобы восстановить дерево циклов программы, которая будет обрабатывать эти данные. То есть, с определенными оговорками можно утверждать следующее: “Используя структуру данных можно построить структуру программы”. Эта идея и есть концептуальной основой метода проектирования программ по структуре данных [3 – 7]. Идея и сущность

* Входная лента это прототип или идеальный образ любого машиночитаемого носителя информации с последовательным доступом. Это также прототип потока любого вида транзакций, сообщений или входных сигналов.

* Этот термин не является атрибутом DS-теории.

этого метода есть частный случай или частный результат *DS*-теории.

Любой канонический алгоритм может быть реализован в виде прикладной программы, но далеко не каждая реально существующая прикладная программа является реализацией канонического алгоритма. Если попытаться снять с какой-либо реальной прикладной программы все адаптационные слои, то может оказаться, что программа реализует, либо только часть канонического алгоритма, либо только некоторую его компоненту. Возможна также и другая ситуация. После освобождения канонического алгоритма от адаптационных слоев может оказаться, что канонический алгоритм является комбинацией нескольких более простых канонических алгоритмов.

Операции над алгоритмами

В *DS*-теории исследуются различные манипуляции или операции со схемами декомпозиции, которые в определенных условиях являются исчислением алгоритмов. Ниже приведено короткое описание некоторых из этих операций над схемами.

1. Дополнение *DS* (рис. 7).

Эта операция значит, что в процессе исследования (проектирования) объекта есть необходимость сделать более глубокую декомпозицию. Возможная причина – в результате декомпозиции будет достигнута более высокая точность расчетов.

В результате этой операции заменен один фрагмент КС на другой, вследствие чего КС стала более длинной:

КС была: $f_1 \to g_1 \to h_1 \to f_2 \to g_2 \to h_2$.

КС стала: $f_1 \to g_1 \to h_1 \to f_2 \to l_1 \to m_1 \to n_1 \to h_2$.

В дереве декомпозиции появилось больше узлов и больше уровней.

2. Усечение *DS* (рис. 8).

Эта операция значит, что в уже существующей схеме декомпозиции в узле 3 между P-свойствами найдена приемлемая аналитическая зависимость g_6, которая позволяет сделать расчет другим образом, и нет необходимости делать более глубокую декомпозицию.

В результате этой операции заменен один фрагмент КС на другой, вследствие чего КС стала короче:

КС была: $f_1 \to g_1 \to h_1 \to f_2 \to f_5 \to g_5 \to h_5 \to h_2 \to f_3 \to g_3 \to h_3$.

КС стала: $f_1 \to g_1 \to h_1 \to f_2 \to g_6 \to h_2 \to f_3 \to g_3 \to h_3$.

3. Синтез *DS* (рис. 9).

Эта операция значит, что в процессе расчета по КС в первом дереве возникла необходимость одновременно выполнить ещё одну расчетную процедуру.

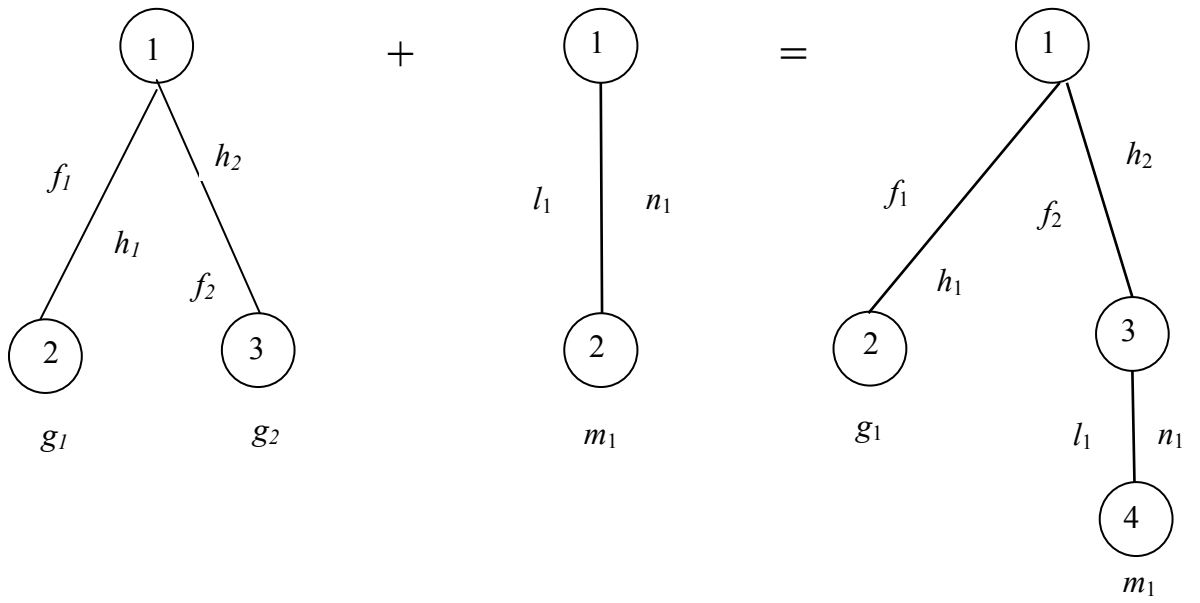


Рис. 7. Дополнение дерева *DS*

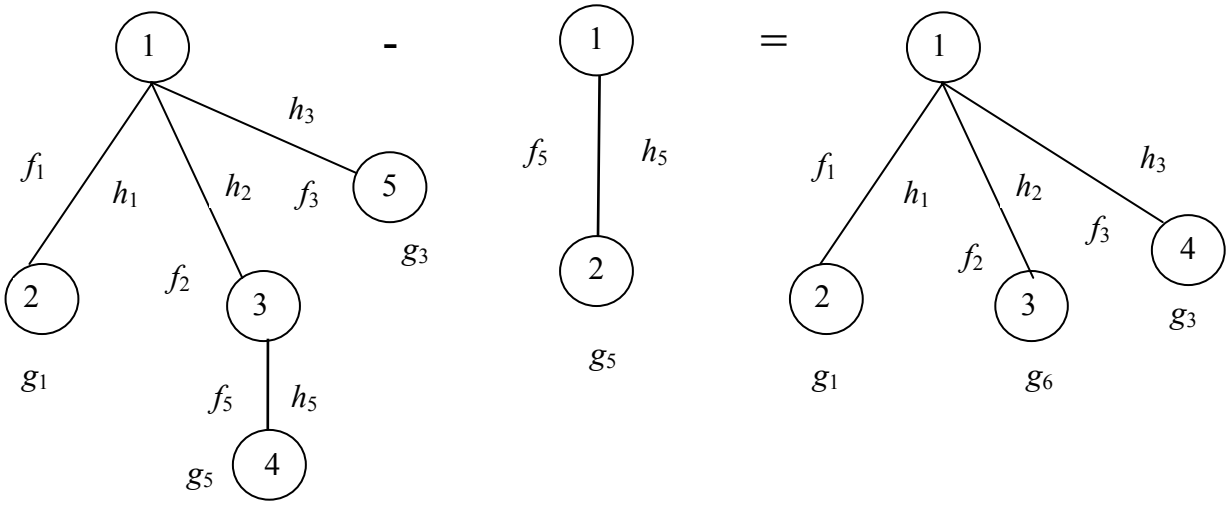


Рис. 8. Усечение *DS*

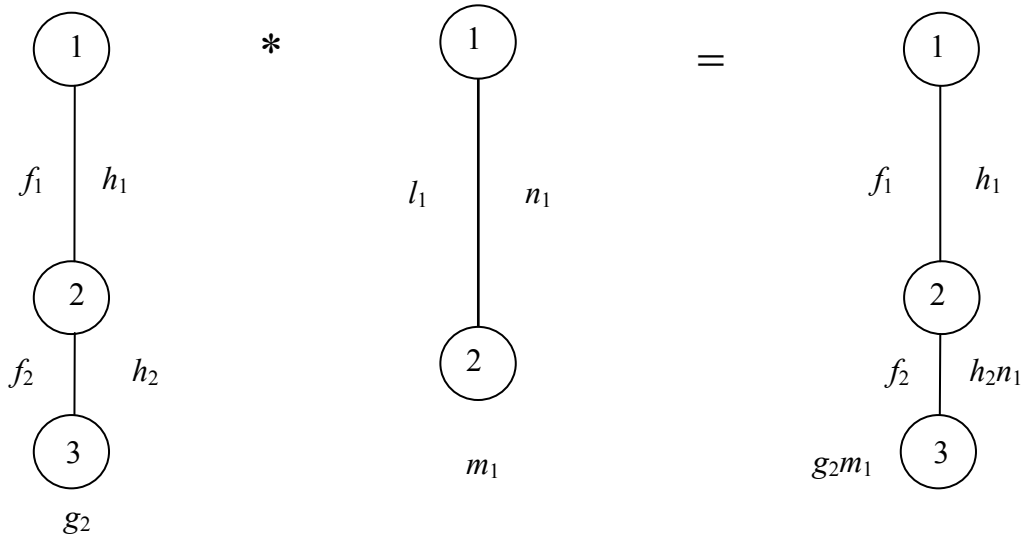


Рис. 9. Синтез *DS* (вариант б)

В этой операции должно выполняться условие: результаты декомпозиции (количество частей) по схемам f_2 и l_1, должны быть одинаковы.

Операция имеет два варианта:

а) аналитические зависимости g_1 и m_1 каким-то образом связаны. Тогда в КС появится суперпозиция этих зависимостей g_2 α (m_1) или m_1 β (g_2).

КС была: f_1 -› f_2 -› g_2 -› h_2 -› h_1.

КС стала: f_1 -› f_2 -› g_2 α (m_1) {или m_1 β (g_2)} -› (h_2, n_1) -› h_1.

б) аналитические зависимости g_1 и m_1 не связаны одна с другой. Тогда в КС они просто будут перечислены рядом в произвольном порядке.

Исходная КС была: f_1 -› f_2 -› g_2 -› h_2 -› h_1.

Дополнительная КС есть: l_1 -› m_1 -› n_1 .

КС как результат синтеза стала: f_1 -› f_2 -› (g_2, m_1) -› (h_2, n_1) -› h_1 .

Генерация алгоритмов

DS-теория создает предпосылки для генерации алгоритмов (не машинного кода). Исходными данными для генерации выступает схема декомпозиции, соответствующий ей канонический алгоритм, а также кортежи описания узлов и кортежи описаний алгоритмически релевантных факторов. Каждый из кортежей соотносится с узлом дерева *DS*. Сгенерированный алгоритм может быть использован как исходные данные для генерации машинного кода.

Направление исследований с точки зрения *DS*-теории

В настоящее время есть несколько направлений дальнейшего развития *DS*-теории. Первое – изучение типологии схем декомпозиции и их компонент. Второе – это исследование алгоритмически релевантных факторов и анализ их влияния на канонический алгоритм. Третье – проектирование процедур генерации алгоритмов. В каждом из направлений продвижение может быть планомерным, прогнозируемым и обозримым. Планомерность исследований реальна, несмотря на то, что совокупность всех реальных алгоритмов и программ, существующих в настоящее время, во всех прикладных областях может показаться необозримым. Объясняется это следующим образом.

С точки зрения *DS*-теории множество алгоритмов означает множество схем декомпозиции представленных деревом. Каждая схема имеет произвольное количество ребер и узлов. Алгоритмически релевантные факторы влияют на алгоритмы одного узла и (или) на смежные с ним – предшественник или потомок. Алгоритмически релевантных факторов ограниченное количество. Их влияние на узлы практически одинаково, независимо от того, где в схеме находится узел. Также ограничено количество типов узлов. То есть, исследуются и анализируются только узлы дерева и их модификации. А количество узлов, их типов и их модификаций существенно меньше, по сравнению с количеством деревьев, составленных из этих узлов. То есть, *DS*-теория позволяет изучать вместо неограниченного количества реально и потенциально существующего множества алгоритмов ограниченное количество алгоритмических объектов, значительно более простых, чем реальные прикладные алгоритмы. *DS*-теория позволяет также проектировать эти конструкции и манипулировать ими не интуитивно, а логически обосновано, даже рутинно.

Выводы

В статье описана схема декомпозиции и ее атрибуты. Введены понятия квалификационного предложения, свойства-совокупности, алгоритмической зависимости и контура синтеза. Показана алгоритмическая природа схемы. Схема декомпозиции рассматривается как основное понятие *DS*-теории. *DS*-теория рассматривается как прототип теории алгоритмов для современного программного обеспечения. Вводится понятие канонического алгоритма как одной из теоретических моделей *DS*-теории. Описываются операции над схемами декомпозиции. Показана возможность использовать при проектировании алгоритмов древовидные двумерные графы вместо строчной буквенной записи. Кратко описано существование различия между каноническим алгоритмом и реальными прикладными алгоритмами.

1. *Ivar Jacobson and Bertrand Meyer.* Methods Need Theory. Dr. Dobb's Journal, 6 August 2009.
2. *Victor R. Basili.* The role of experimentation in software engineering: past, current, and future. Proceedings of the Eighteenth International Conference on Software Engineering (ICSE), Berlin, Germany, March 1996.
3. *Jackson, M.A.* Principles of Program Design. New York: Academic Press, 1975.
4. *Warnier J. –D.* Logical Construction of Programs. Leiden: H.E. Stenfert Kroese B.V., 1975.

5. *Вельбицкий И.В.* Технология программирования. – Киев: Техніка, 1984. – 250 с.
6. *Колесник В.Г.* Совместная обработка файлов и иерархических баз данных в условиях применения метода Джексона. – Донецк: ИЭП АН УССР, 1988. – 16 с.
7. *Колесник В.Г.* Концептуальная основа метода программирования по структуре данных (знаний). – Краматорск, 1986. – 22 с. Деп. В УКРНИИНТИ 6.06.88, № 1409.
8. *Edmund M. Clarke, Jeannette M. Wing, Et Al.* Formal Methods: State of the Art and Future Directions // ACM Computing Surveys. – 1996.– V. 28, N 28.– P. 626– 643.
9. *Бэкус Дж.* Можно ли освободить программирование от стиля фон Неймана? Функциональный стиль и соответствующая алгебра программ. – Лекции лауреатов премии Тьюринга: пер. с англ. / Под ред. Р. Эшенхерста. М.: Мир, 1993. – С. 84– 158.
10. *Марка Д., Макгоуэн К.* Методология структурного анализа и проектирования (SADT), Пер. с англ., МетаТехнология. –.М., 1993. – 240 с.
11. *Brooks Frederick P.*, No Silver Bullet: Essence and Accidents of Software Engineering // Computer. – 1987. – V. 20., N. 4.– P. 10– 19.

Получено 26.04.2011

Об авторе:

Колесник Валерий Георгиевич,
старший научный сотрудник
кафедры АПП.

Место работы автора:

Донбасская государственная
машиностроительная академия.
г. Краматорск, ул. Шкадинова, 72, п/я 13.